Nettoyage perpétuel de réseaux[†]

Lélia Blin[1] and Janna Burman[2] and Nicolas Nisse[3]

[1] *Univ. d'Evry Val d'Essone, et LIP6, Univ. Pierre et Marie Curie, Paris, France*
[2] *Grand Large, INRIA Saclay, et LRI, Univ. Paris-Sud, Orsay, France*
[3] *Mascotte, INRIA, I3S(CNRS/UNS), Sophia Antipolis, France*

Dans le cadre du *nettoyage de graphes contaminés* (*graph searching*), des agents mobiles se déplacent successivement le long des arêtes du graphe afin de les *nettoyer*. Le but général est le nettoyage en utilisant le moins d'agents possible. Nous plaçons notre étude dans le modèle de calcul distribué *CORDA minimaliste*. Ce modèle est muni d'hypothèses très faibles : les nœuds du réseau et les agents sont anonymes, n'ont pas de mémoire du passé ni sens commun de l'orientation et agissent par *cycles Voir-Calculer-Agir* de manière *asynchrone*. Un intérêt de ce modèle vient du fait que si le nettoyage peut être fait à partir de positions arbitraires des agents (par exemple, après pannes ou recontamination), l'absence de mémoire implique un nettoyage perpétuel et donc fournit une première approche de nettoyage de graphe *tolérant aux pannes*. Les contraintes dues au modèle *CORDA minimaliste* nous amènent à définir une nouvelle variante de nettoyage de graphes - le *nettoyage sans collision*, autrement dit, plusieurs agents ne peuvent occuper simultanément un même sommet.

Nous montrons que, dans un contexte *centralisé*, cette variante ne satisfait pas certaines propriétés classiques de nettoyage comme par exemple la monotonie. Nous montrons qu'interdire les "collisions" peut augmenter le nombre d'agents nécessaires d'un facteur au plus Δ le degré maximum du graphe et nous illustrons cette borne. De plus, nous caractérisons complètement le nettoyage sans collision dans les arbres.

Dans le contexte *distribué*, la question qui se pose est la suivante. Existe-t-il un algorithme qui, étant donné un ensemble d'agents mobiles arbitrairement répartis sur des sommets distincts d'un réseau, permet aux agents de nettoyer perpétuellement le graphe ? Dans le cas des chemins, nous montrons que la réponse est négative si le nombre d'agents est pair dans un chemin d'ordre impair, ou si il y a au plus deux agents dans un chemin d'ordre au moins 3. Nous proposons un algorithme qui nettoie les chemins dans tous les cas restants, ainsi qu'un algorithme pour nettoyer les arbres lorsqu'un nombre suffisant d'agents est disponible initialement.

Keywords: algorithme distribué, CORDA, nettoyage de graphe, auto-stabilisation

1 Introduction

Dans le modèle de calcul distribué CORDA (voir, e.g., [Pre01]), chaque entité (agent mobile) est *autonome* et agit par *cycles asynchrones*. Un cycle est constitué de 3 *phases* : *Voir*, *Calculer* et *Agir*. Chaque phase est exécutée de manière *atomique* et *asynchrone*. Une entité a d'abord accès à *vue*, du réseau, comprenant la topologie du réseau et les positions des autres entités dans le réseau à un moment donné (Phase *Voir*). Sur la base de ces informations, l'entité peut effectuer un calcul (Phase *Calculer*) et en déduire une action : se déplacer ou non sur un sommet voisin (Phase *Agir*). La difficulté provient de ce que les entités autonomes doivent arriver à un but commun et donc doivent se coordonner sans communication autre que les informations apportées par la vue du réseau. En particulier, du fait de l'asynchronie, la phase de calcul et celle d'action peuvent être basées sur une vue obsolète du réseau. De nombreux travaux étudient l'impact d'hypothèses différentes dans ce modèle (e.g., [FIPS10, CFMS10]). Dans ce travail, nous considérons le modèle, que nous appelons *CORDA minimaliste*, avec des hypothèses très faibles. Le graphe (nœuds et liens) est *anonyme* et aucune mémoire locale n'est disponible sur les nœuds. Les agents sont *anonymes*, *uniformes*, n'ont pas de mémoire du passé, ni de sens commun de l'orientation. De plus, plusieurs agents ne peuvent pas occuper simultanément un même sommet (*contrainte d' exclusivité*). Toutes ces hypothèses

[†]Les nombreux détails omis dans cette note peuvent être consultés dans [BBN12].

font apparaitre des symétries qu'il est difficiles de briser, ce qui rend la conception d'algorithmes plus difficile. Les algorithmes proposés dans ce modèle sont *auto-stabilisant* [Dij74] : à partir d'une configuration quelconque [‡], un algorithme atteint une *configuration légale*, i.e., toutes les arêtes sont *propres*, et reste propres. Par conséquent, les algorithmes distribués proposés résistent à la *recontamination*.

État de l'art. Récemment, plusieurs problèmes impliquant des agents mobiles se déplaçant le long des arêtes du réseau ont été étudiés dans le modèle CORDA. Dans le problème de l'*exploration avec arrêt*, on veut que chaque sommet soit visité au moins une fois par un agent [FIPS10, CFMS10]. Dans celui de l'*exploration perpétuelle*, chaque sommet doit être visité au moins une fois par *chaque* agent [BMPBT10, BMPBT11]. Un autre problème classique est le *rendez-vous* où tous les agents doivent se retrouver sur un même sommet [DSN11]. Nous plaçons notre étude dans le même cadre que [BMPBT10, DSN11, BMPBT11], c'est-à-dire le modèle CORDA minimaliste, donc exclusif.

Le *nettoyage de graphe* a été introduit par Breisch en 1967 [Bre12] puis formalisé par Parsons [Par78]. Il a ensuite été largement étudié pour son lien avec les largeurs arborescente (*treewidth*) et linéaire (*pathwidth*) des graphes [FT08]. Soit G un graphe dont les arêtes sont initialement *contaminées*, une *stratégie de nettoyage* est une séquence d'actions qui aboutit au *nettoyage* de toutes les arêtes. Les actions possibles de nettoyage "classique" sont (1) placer un agent sur un sommet de G, (2) supprimer un agent d'un sommet de G et (3) déplacer un agent le long d'une arête. Une arête est *nettoyée* si (A) un agent la traverse et/ou (selon les variantes) (B) ses deux extrémités sont occupées. Une arête e est *recontaminée* sitôt qu'il existe un chemin sans agents de e à une arête contaminée. Le nombre d'agents utilisés par une stratégie est le nombre maximum d'agents occupant les sommets de G au cours de la stratégie. Plusieurs variantes de nettoyage, combinants différement les actions $1,2,3$ et les façons de nettoyer A,B, ont été étudiées mais s'avèrent équivalentes [FT08]. L'*encerclement* d'un graphe G, noté $ns(G)$, est le nombre minimum d'agents nécessaires pour nettoyer un graphe G dans la variante $1,2+B$ (dite *node-search*).

Nettoyage sans collision. Dans le modèle CORDA, il est naturel de n'autoriser que l'action 3, les agents ne pouvant "sauter" d'un sommet à un autre. Par ailleurs, si deux (ou plus) agents pouvaient occuper le même sommet, l'asynchronie du modèle pourrait faire en sorte qu'ils agissent toujours comme un unique agent. C'est pourquoi nous imposons la contrainte d'exclusivité. Cependant, avec ces contraintes, il n'est pas possible de nettoyer un anneau, quel que soit le nombre d'agents utilisés, si l'on n'autorise le nettoyage que grâce à la façon A. Ainsi, dans notre modèle, le nettoyage peut se faire de la façon A ou B.

Résultats. Dans un premier temps, nous étudions une variante *centralisée* [§] de nettoyage sans collision (Sec. 2). Cette étude nous permet d'évaluer les performances (en termes de nombre d'agents) des algorithmes distribués présentés dans la Sec. 3 et de trouver des techniques de nettoyage distribué. Cette étude est intéressante par elle-même car cette nouvelle variante de nettoyage s'avère satisfaire des propriétés très différentes des variantes "classiques" de nettoyage. Dans notre cas, les stratégies s'apparentent à celles d'un jeu de taquin. Dans un second temps, nous étudions le nettoyage sans collision dans le modèle CORDA minimaliste (Sec. 3). Nous caractérisons complètement ce problème dans le cas des réseaux en chemins [¶] et nous proposons un algorithme de nettoyage perpétuel des arbres.

2 Étude du nettoyage sans collision en centralisé

Nous définissons une stratégie de *nettoyage sans collision* d'un graphe $G=(V,E)$ avec $k \geq 1$ agents par (a) un ensemble $I \subseteq V$ de k sommets qui sont les positions initiales distinctes des agents, et (b) une séquence des actions suivantes : un agent en $u \in V$ peut se déplacer sur un de ses voisins $v \in N(v)$ si v est inoccupé. Il doit résulter d'une telle stratégie le nettoyage de toutes les arêtes soit (A) en étant traversées, soit (B) en ayant leurs deux extrémités occupées.

Les variantes "classiques" de nettoyage (e.g., node-search) satisfont deux propriétés essentielles qui les

‡. Ici, "quelconque" signifie "arbitraire mais satisfaisant la contrainte d'exclusivité" puisque cette contrainte fait partie du modèle. Sans cette contrainte, le nettoyage ne peut pas être réalisé dans le modèle CORDA en partant d'une configuration arbitraire. Notons que, dans le cas du nettoyage de graphe, une configuration inclut l'ensemble des arêtes propres en plus de la position des agents.

§. Par opposition au cas distribué (cf., Sec. 3), dans le cas centralisé, une entité externe contrôle les mouvements des agents.

¶. Notons que dans le cas d'un chemin, l'exploration perpétuelle implique le nettoyage. L'inverse n'est pas vrai. Par ailleurs, dans un chemin avec ≥ 3 sommets, l'exploration perpétuelle sans collision est clairement impossible dans le modèle CORDA minimaliste.

rendent plus "simples" à étudier. Elles sont closes par mineur ‖ et sont monotones **. Nous prouvons que le nettoyage sans collision n'est ni monotone ni clos par sous-graphes (induits ou non). Par ailleurs, ajouter la contrainte d'exclusivité peut augmenter arbitrairement le nombre d'agents. Par exemple, soit G une étoile avec $\Delta \geq 3$ feuilles, alors $ns(G) = 2$ alors que $ws(G) = \Delta - 1$, où $ws(G)$ est le nombre minimum d'agents nécessaires au nettoyage sans collisions du graphe G. Nous prouvons que :

Théorème 1. *Pour tout graphe G de degré maximum Δ, soit $f(\Delta) = \Delta - 1$ si $\Delta > 3$ et $f(\Delta) = 1$ sinon, $ns(G) - 1 \leq ws(G) \leq f(\Delta) \cdot ns(G)$*

Le théorème 1 montre qu'un algorithme polynomial pour calculer ws dans la classe des graphes de degré ≤ 3 serait une approximation à une constante près de ns et donc de la pathwidth. Le calcul de la pathwidth est NP-complet dans cette classe de graphes et non-approximable à une constante près en général.

La propriété de clôture par sous-graphe n'est pas vraie pour le nettoyage sans collision : soit G le graphe obtenu à partir de $k > 2$ triangles en identifiant un sommet de chacun en un unique sommet, et soit H une étoile à $2k$ feuilles. H est un sous graphe de G et $2k - 1 = ws(H) > ws(G) = k + 1$. Dans le cas des arbres, cette propriété de clôture par sous-graphe reste cependant vraie. Ce résultat n'est pas aussi trivial que dans le cas "classique" puisque transformer une stratégie de T en une stratégie dans un sous-arbre S nécessite de savoir où placer tous les agents dans S sans qu'ils ne se rencontrent (en satisfaisant la contrainte d'exclusivité), tout en s'assurant qu'il soit possible de les amener à l'endroit désiré lorsqu'on doit les utiliser.

Théorème 2. *Pour tout arbre T et sous arbre S de T, $ws(S) \leq ws(T)$.*

Le théorème 2 nous permet d'adapter la caractérisation de l'encerclement "classique" dans le cas des arbres due à Parson [Par78]. Ainsi, le calcul de $ws(T)$ est polynomial (par programmation dynamique).

Théorème 3. *Pour tout arbre T et $k \geq 1$, $ws(T) \leq k$ si et seulement si le degré maximum de T est $\leq k+1$, et pour tout sommet $v \in V(T)$, pour toute composante connexe S de $T \setminus \{v\}$ (ou* branche en v*), $ws(S) \leq k$ et $\forall i \geq 2$ pair, $|\{S$ branche en $v : \ ws(S) \geq k - i/2 + 1\}| \leq i$.*

3 Nettoyage perpétuel de graphe

Dans cette section, nous considérons le problème de nettoyage sans collision dans le modèle de calcul distribué CORDA minimaliste défini dans l'introduction. Les algorithmes distribués désirés doivent nettoyer les graphes à partir d'une configuration quelconque satisfaisant la propriété d'exclusivité. Dans ce contexte, l'algorithme (uniforme) exécuté par chaque agent autonome prend une vue (positions des agents) en entrée et doit décider, malgré l'asynchronie, si l'agent peut se déplacer ou non, et où il doit aller.

Théorème 4. *Soit P un chemin de n sommets et $1 \leq k \leq n$. Alors il existe un algorithme de nettoyage perpétuel de P avec k agents si et seulement si $(1 \leq k \leq n \leq 2)$ OU $((3 \leq k \leq n)$ ET (n pair OU k impair)).*

Idée de la preuve. Supposons que $n \geq 3$ est impair et k pair. Dans ce cas, dans n'importe quelle configuration de depart symétrique (même satisfaisant l'exclusivité), le sommet central c est inoccupé. De plus, la symétrie peut faire que quel que soit l'algorithme utilisé, la symétrie de la position des agents est conservée (si un agent bouge, l'agent symétrique a la même vue du réseau et donc choisit de se déplacer à l'identique). Ainsi, lorsque c est occupé (il doit l'être pour nettoyer le chemin), il l'est par deux agents, contredisant la propriété d'exclusivité. Ainsi, un chemin d'ordre impair ne peut être nettoyé par un nombre pair d'agents à partir d'une configuration arbitraire.Considérons maintenant le cas $3 \leq k < n$ et $k = 2r' + 1$ impair. Nous montrons qu'il existe un algorithme qui, à partir de n'importe quelle configuration avec exclusivité, permet aux agents d'atteindre la configuration A, Fig. 1. Une fois dans cette configuration, l'algorithme est celui représenté sur la Fig. 1. Sur cette figure, un rectangle avec x dedans représente x sommets consécutifs et occupés. Un carré vide représente un sommet occupé, et un rond un sommet inoccupé. Un segment avec y au dessus représente y sommets inoccupés consécutifs. Une flèche rouge représente le mouvement (unique) à réaliser dans la configuration correspondante. Enfin, les flèches noires (pointillées ou non) représentent la succession des configurations. L'algorithme dans le cas n et k pairs peut être consulté dans [BBN12]. □

‖. Sans entrer dans les détails, cela signifie entre autre que $ns(H) \leq ns(G)$ pour tout sous-graphe H de G.

**. Dans le cas du node-search, cela siginifie que, pour tout graphe G, il existe une stratégie de nettoyage de G avec $ns(G)$ agents telle qu'une arête n'est jamais recontaminée une fois qu'elle a été nettoyée.

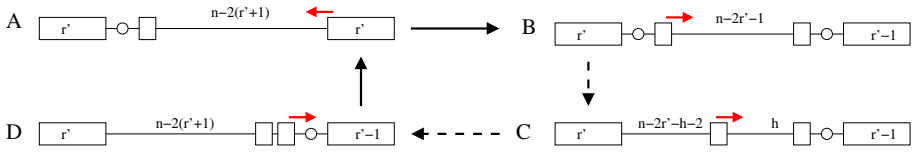

FIGURE 1: Stratégie de nettoyage perpétuel d'un chemin de n sommets avec $k = 2r' + 1 \geq 3$ agents.

Dans le cas d'un arbre quelconque T, avec $x \geq 2$ feuilles adjacentes au même sommet v. Si dans la configuration initiale, $2 \leq y \leq x$ de ces feuilles sont occupées, les y agents correspondants ne peuvent plus bouger, et ceci quel que soit l'algorithme suivi. En effet, si l'un d'entre eux se déplace sur v, tous peuvent faire de même par symmétrie et asynchronisme, et la "collision" ne peut être évitée sur v. Nous généralisons ce fait. La *vue-topologique* d'un agent en $v \in V(T)$ est l'ensemble des branches en v. Un sommet $v \in V(T)$ est un *b-node* si il existe un sommet $u \in V$ tel que u et v ont la même vue-topologique et le chemin entre u et v a un nombre impair de sommets (par exemple, si u et v sont 2 feuilles adjacentes à un même sommet).

Théorème 5. *Soit T un arbre sans b-nodes, alors il existe un algorithme de nettoyage perpétuel (dans le modèle CORDA minimaliste) de T avec $k = O(ws(T))$ agents.*

Idée de la preuve. Supposons de plus que T n'est pas symétrique (il n'y a pas d'arête e telle que les deux arbres de $T \setminus \{e\}$ sont isomorphiques). Dans ce cas, il est possible d'assigner à chaque sommet de T un identifiant unique. Ainsi, il est possible d'exécuter l'algorithme centralisé (cf. preuve du Th. 3) sans ambiguïté (lorsque plusieurs mouvements sont possibles, l'agent prioritaire est celui sur le sommet de plus petit identifiant). $O(1)$ agents supplémentaires (par rapport à $O(ws(T))$) sont utilisés pour casser la symétrie de la stratégie. Si T est symétrique, soient T_1 et T_2 (isomorphiques) les deux "côtés" de T, notre algorithme considère séparement ces 2 sous-arbres en utilisant $O(ws(T_1)) + O(ws(T_2)) = O(ws(T))$ agents plus $O(1)$ agents supplémentaires ($O(1)$ par côté) pour casser la symétrie. □

Théorème 6. *Soit T un arbre avec B l'ensemble de ses b-nodes, $|B| = b$, et l'arbre $T_0 = T \setminus B$, alors il existe un algorithme de nettoyage perpétuel de T avec $b + O(ws(T_0))$ agents et $\exists X$, $\{k \text{ pair } : k \leq b\} \subseteq X \subseteq \{k : k < b + O(ws(T_0))\}$, tel que $\forall k \in X$, aucun tel algorithme ne peut nettoyer T avec k agents.*

Idée de la preuve. Pour la 1^{re} assertion, nous proposons un algorithme qui commence par déplacer b agents sur les sommets de $V(T) \setminus V(T_0)$. Puis les $O(ws(T_0))$ agents restants nettoient T_0 (Th. 5). □

Références

[BBN12] L. Blin, J. Burman, and N. Nisse. Perpetual graph searching, 2012. En cours de rédaction. http://www-sop.inria.fr/members/Nicolas.Nisse/perpetual.pdf.

[BMPBT10] L. Blin, A. Milani, M. Potop-Butucaru, and S. Tixeuil. Exclusive perpetual ring exploration without chirality. In *24th International Symposium on Distributed Computing (DISC)*, volume 6343, pages 312–327. Springer, 2010.

[BMPBT11] François Bonnet, Alessia Milani, Maria Potop-Butucaru, and Sébastien Tixeuil. Asynchronous exclusive perpetual grid exploration without sense of direction. In *OPODIS*, pages 251–265, 2011.

[Bre12] R.L. Breisch. *Lost in a Cave-applying graph theory to cave exploration*. 2012.

[CFMS10] J. Chalopin, P. Flocchini, B. Mans, and N. Santoro. Network exploration by silent and oblivious robots. In *36th International Workshop on Graph Theoretic Concepts in Computer Science (WG)*, volume 6410, pages 208–219, 2010.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11) :643–644, 1974.

[DSN11] G. D'Angelo, G. Di Stefano, and A. Navarra. Gathering of six robots on anonymous symmetric rings. In *SIROCCO*, pages 174–185, 2011.

[FIPS10] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Remembering without memory : Tree exploration by asynchronous oblivious robots. *Theor. Comput. Sci.*, 411(14-15) :1583–1598, 2010.

[FT08] F.V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3) :236–245, 2008.

[Par78] T. D. Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, volume 642, pages 426–441. 1978.

[Pre01] G. Prencipe. Instantaneous actions vs. full asynchronicity : Controlling and coordinating a set of autonomous mobile robots. In *ICTCS*, pages 154–171, 2001.